Introduction to control systems
TI0118

Homework 2

Student name: Kelvin Leandro Martins
Student number: 540006
Student name: Pedro Leinos Falcão Cunha
Student number: 542114

Exercise 1

The unit steps responses of two systems A and B are recorded and reported in the files attached to the homework assignment. In each file, the first column gives the time vector, t, and the second column provides the output response, $y(t)$, for the systems A (in `HW2_ex1_dataA.txt`) and B (in `HW2_ex1_dataB.txt`). Do the following:

1. Load the data and plot the responses for systems A and B.
2. Identify the order of the systems. Based on the plots, estimate the transient response characteristics, such as time constant, settling time, rise time, peak time and percentage of overshoot. Write the corresponding transfer functions $T_A(s)$ and $T_B(s)$ for the systems A and B.
3. In the plot obtained in item (1) of this exercise, add and compare the unit step response of the systems $T_A(s)$ with the data provided in `HW2_ex1_dataA.csv`. Do the same with the unit step response of $T_B(s)$ with the data provided in `HW2_ex1_dataB.csv`. Comment on your results.

Solution

Theoretical background

The total response of the system is given by the sum of the zero-input response and the zero-state response. The *zero-input response* (or free response) is the system's response when $x(t) = 0$, which means that the system is in its initial state. So this is result of the internal conditions of the system alone, or in other words, it does not depend on the input. The *zero-state response* (or forced response) is the output of a system when the initial conditions are set to zero and the system is stimulated by an input signal. This response shows how the system reacts to external inputs.

Having a transfer function *G(s)*, we can determine the roots and poles of that given system. According to [Nise, 2011], the poles are the roots of the denominator, or in other words, the values for s that make the transfer function infinite. The zeros, on other hand, are the roots of the numerator.

A transfer function of a first-order system typically has the following format:

$$G(s) = \frac{K}{\tau s + 1} \tag{1}$$

The value K is referred to as the system gain (or steady-state gain), and τ is the time constant, which indicates the speed at which the system responds to changes and represents the time required for the step response to reach 63% of its steady-state value. A small τ signifies a rapid response, while a large τ indicates a slower response.

In addition to the time constant, other transient response specifications include:

- Rise Time (T_r): The time required for the system to transition from 10% to 90% of its final value.
- Settling Time (T_s): The duration needed for the system to reach and remain within 2% of its final value.

Second-order systems may provide more variable and complex shapes of responses compared to first-order systems. The second-order transfer function has the general form:

$$G(s) = \frac{b}{s^2 + as + b} \tag{2}$$

and the Figure 1 illustrates some examples of these diverse response characteristics.
Two important variables in second-order systems are called natural frequency (w_n) and damping ratio (ζ). The *natural frequency* is the rate at which a system oscillates when not subjected to external forces. The *damping ratio* is a measure that describes how quickly the oscillations decay from one bounce to the next. The damping ratio can be defined as:

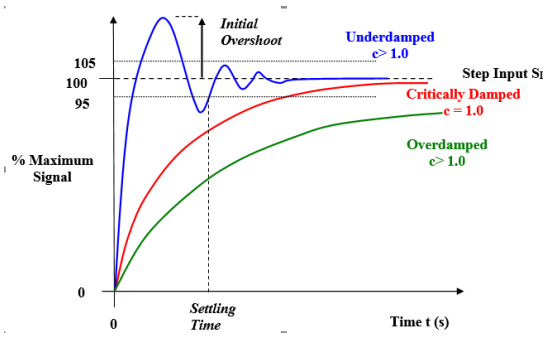

Figure 1: Second-order system responses.

$$\zeta = \frac{1}{2\pi} \frac{\text{Natural period (seconds)}}{\text{Exponential time constant}}$$

Using ζ and w_n, we can transform the Equation 2 in terms of these two variables as follow:

$$G(s) = \frac{w_n^2}{s^2 + 2\zeta w_n s + w_n^2} \tag{3}$$

Now let's explain each response type:

- Undamped: For $\zeta = 0$, the system experiences no damping, leading to continuous oscillations around the steady-state value. The amplitude of these oscillations remains constant over time, and the system never settles at a specific value.
- Underdamped: For $0 < \zeta < 1$, the system oscillates around the steady-state value before gradually settling. The oscillations diminish over time, and the system takes longer to reach stability compared to critically damped systems.
- Critically damped: With $\zeta = 1$, the system returns to equilibrium as quickly as possible without oscillating. It achieves the fastest possible response time to reach and stay at the steady-state value.
- Overdamped: When the damping ratio $\zeta > 1$, the system's response is slow and does not oscillate. It approaches the steady-state value smoothly but with a longer settling time compared to critically damped systems.

In addition to the transient response specifications explained for first-order systems, the underdamped second-order system introduces two important specifications:

- Peak time (T_p): The time required for the system to reach its maximum peak during the transient response.

- Percent overshoot (%OS): A measure of how much the system's output exceeds its final steady-state value during its transient response. The percent overshoot, $\%OS$, is given by:

$$\%OS = \frac{c_{max} - c_{final}}{c_{final}} \times 100\% \tag{4}$$

  Here, c_{max} is the system's output evaluated at the peak time T_p, and c_{final} represents the system's steady-state value.

The underdamped second-order system, which is a common model for many physical problems, exhibits unique behaviors that are essential for both analysis and design. To better understand and utilize these behaviors, we need to define the transient specifications associated with underdamped responses.

These transient specifications, such as peak time and percent overshoot, can be directly related to the system's pole locations. By understanding the relationship between pole location and the system's response, we can connect these locations to specific system parameters. This connection allows us to design systems that achieve the desired response by appropriately configuring the system components.

Item 1.1

Figure 2 illustrates the unit step response for data A.

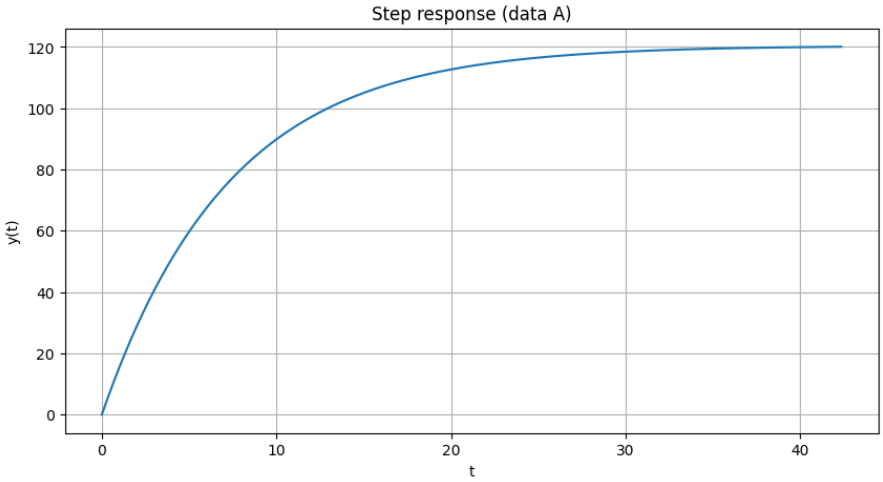

Figure 2: unit step response for system A.

Figure 3 illustrates the unit step response for data B.

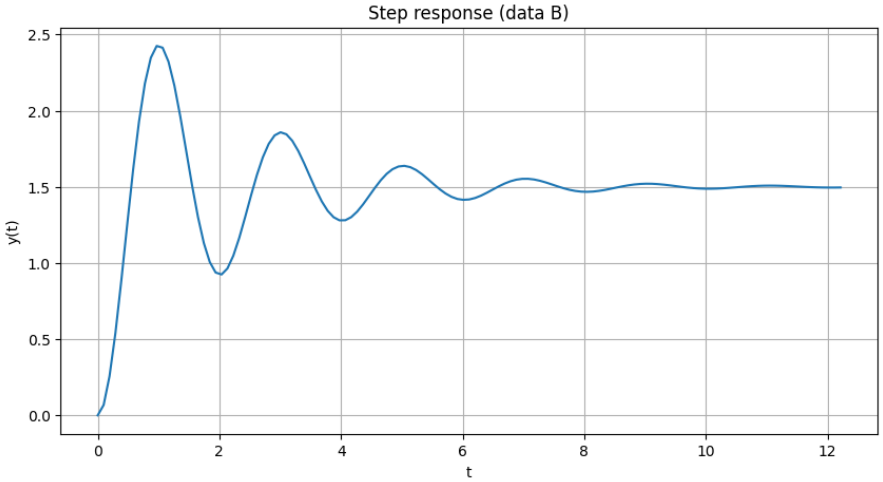

Figure 3: unit step response for system B.

Item 1.2

Initially, figure 2 represents a first order system. Since it's a first-order system, we can estimate the transient response characteristics of the system, such as time constant, settling time and rise time.

- **Time Constant (τ):** For finding the time constant τ, we have to find the equivalent time of 63.2% of the steady state. According to the plot, the final value of the unit step response in system A is 120.0862. So, the time constant τ is approximately 7.3959 seconds.
- **Rise Time (T_r):** For finding the rise time T_r, we have to find the time required to transition from 10% to 90% of its final value. According to the plot, the final value of the unit step response in system A is 120.0862. So, the rise time T_r is approximately 15.8004 seconds.
- **Settling Time (T_s):** For finding the rise time T_s, we have to find the equivalent time for the system to reach and remain within 2% of its final value. According to the plot, the final value of the unit step response in system A is 120.0862. So, the settling time T_s is approximately 27.5665 seconds.
- **Percent Overshoot ($\%OS$):** For a first-order system, the percent overshoot is always 0%, as the response monotonically approaches the final value without exceeding it. Thus, $\%OS$ is equal to 0%.

With these data, we can write the correspondent transfer function of a first-order system for the unit step response shown in figure 2 using equation (1):

$$G(s) = \frac{120.0862}{7.3959s + 1} \tag{5}$$

Moreover, figure 3 represents a underdamped second-order system. Since it's a underdamped second-order system, we can estimate more transient response characteristics of the system than in the first-order one, such as peak time and percent-overshoot (besides settling time and rise time).

- **Rise Time (T_r):** For finding the rise time T_r, we have to find the time required to transition from 10% to 90% of its final value. According to the plot, the final value of the unit step response in system B is 1.4957. So, the rise time T_r is approximately 0.3878 seconds.
- **Settling Time (T_s):** For finding the rise time T_s, we have to find the equivalent time for the system to reach and remain within 2% of its final value. According to the plot, the final value of the unit step response in system B is 1.4957. So, the settling time T_s is approximately 7.4652 seconds.

- **Peak Time (T_p):** For finding the peak time T_p, we have to find the equivalent time for the system to reach it's maximum value. According to the plot, the maximum value of the unit step response in system B is 2.4248. So, the peak time T_p is approximately 0.9695 seconds.

- **Percent Overshoot ($\%OS$):** For finding the percent overshoot $\%OS$, we have to use equation (4). According to the plot, $c_{max} = 2.4248$ and $c_{final} = 1.4957$ and . So, the percent overshoot $\%OS$ is approximately 62,1181%.

with the percent overshoot $\%OS$ we can find additional information about the system, such as the damping ratio (ζ) and the natural frequency (w_n) of the system.

- **Damping ratio (ζ):** damping ratio (ζ) can also be defined as:

$$\zeta = \frac{-\ln\left(\%OS/100\right)}{\sqrt{\pi^2 + \ln^2\left(\%OS/100\right)}} \tag{6}$$

  In this case, $\zeta \approx 0.14984670976706022$.

- **Natural frequency (w_n):** with damping ratio (ζ) and time peak (T_p), we can also define natural frequency (w_n) as:

$$w_n = \frac{\pi}{T_p \times \sqrt{1 - \zeta^2}} \tag{7}$$

  In this case, $w_n \approx 3.277430310696813$.

With these data, we can write the correspondent transfer function of a second-order system for the unit step response shown in figure 3 using equation (3) (note that this system have a gain of 1.4957):

$$G(s) = \frac{16.0661355}{s^2 + 0.9822243s + 10.74154944}$$

Item 1.3

Figure 4 illustrates the unit step response for data A and system A.

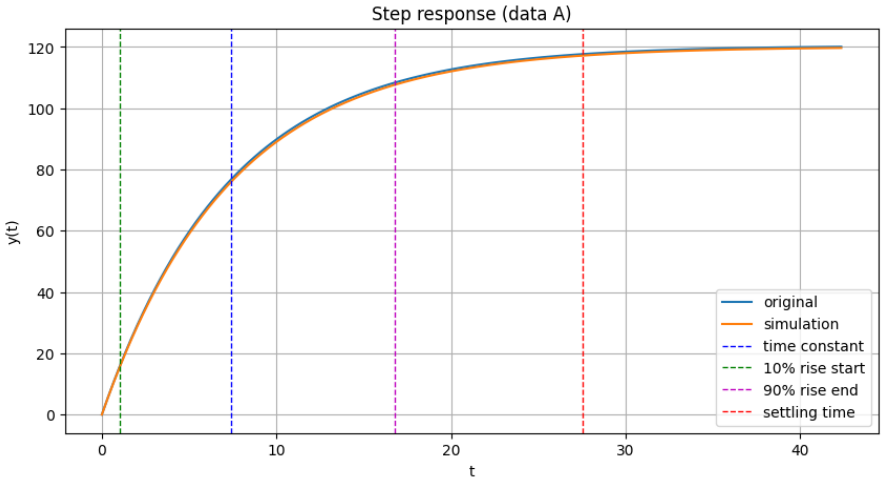

Figure 4: unit step response for data A and system A.

Figure 5 illustrates the unit step response for data B and system B.

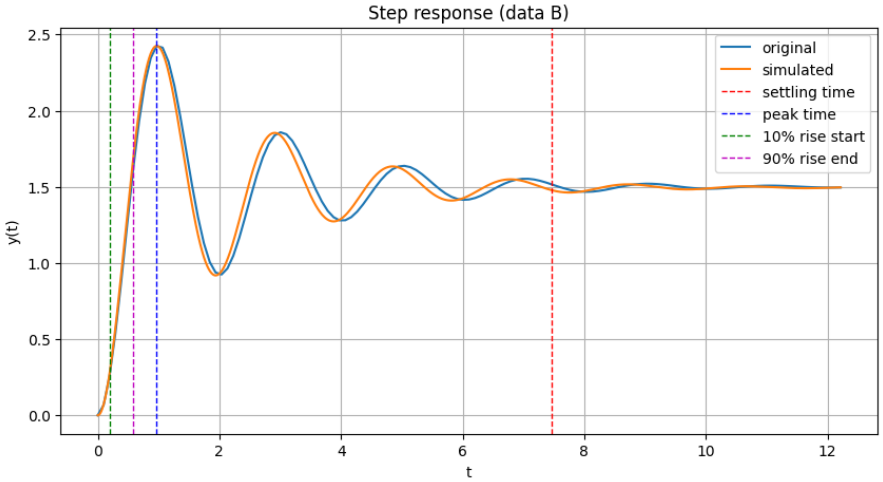

Figure 5: unit step response for data B and system B.

Figure 4 visually confirms that the step response for system A shapes very well data A. Figure 5 also confirms that the step response for system B shapes very well data B, since it's almost perfectly one above other, as in figure 4.

Exercise 2

Consider a system with the transfer function in (8):

$$G(s) = \frac{\tau s + 1}{(0.1s + 1)(0.002s^2 + 0.02s + 1)(s^2 + 0.1s + 1)} \tag{8}$$

Consider two different cases with (i) $\tau = 0.5$ and (ii) $\tau = 20$. For each case separately, do the following:

1. Find the poles and zeros of the system. Plot them in a zero-pole map and draw your conclusions.
2. Use the dominant-poles argument to find an equivalent second-order transfer function. Can the zero be neglected in this case?
3. Plot and compare the step responses for system in equation (8) and the second order equivalent one. Explain and justify the differences between the responses.

Solution

Theoretical background

For systems of order higher than 2, there is no standard set of equations that will lead to the system's response as there is for first and second-order systems, and each waveform will depend on the location of the poles. In some cases, depending on the poles, it may be possible to approximate this higher-order system to a second-order system, depending on the presence of dominant poles.

Dominant poles are the poles that are closest to the $j\omega$-axis. In a system with multiple poles, those closest to the origin are the poles that will produce exponentials with slower decay, causing the final response of the system to be more influenced by these poles.

In cases where there are poles and zeros, it is possible to cancel poles with zeros, even when they are not exactly equal. To do this, we must compare the pole closest to the zero in question by expanding the fraction corresponding to the system's output $C(s)$ in the Laplace domain into partial fractions. The term that remains in the numerator when decomposing into partial fractions is called the residue, and if the residue related to the pole in question has a magnitude significantly different from the others, the zero can be canceled with the pole without significantly affecting the system's time dynamics.

[Ogata, 2010] argues that a pole close to a zero has a small residue, and the coefficient of the corresponding transient term will also be small. A pole far from the origin may have a small residue, causing the transient of this pole to be small and last for a short time. These terms with small residues contribute little to the system's response, allowing them to be neglected and the system to be approximated to one of lower order.

ITEM 2.1

case $\tau = 0.5$:

$$G(s) = \frac{0.5s + 1}{(0.1s + 1)(0.002s^2 + 0.02s + 1)(s^2 + 0.1s + 1)} \tag{9}$$

For equation (9), we have have the following Poles and Zeros:

Poles ($\tau = 0.5$):

$$\begin{aligned}
(\text{root}_1 \text{ of } 0.002s^2 + 0.02s + 1): &\quad -5.00 + 21.79j \\
(\text{root}_2 \text{ of } 0.002s^2 + 0.02s + 1): &\quad -5.00 - 21.79j \\
(\text{root of } 0.1s + 10): &\quad -10.0 + 00.00j \\
(\text{root}_1 \text{ of } s^2 + 0.1s + 1): &\quad -0.05 + 1.000j \\
(\text{root}_2 \text{ of } s^2 + 0.1s + 1): &\quad -0.05 - 1.000j
\end{aligned}$$

Zeros ($\tau = 0.5$):

$$[-2.]$$

Figure 14 illustrates the poles and zeros of the system described by equation (9):

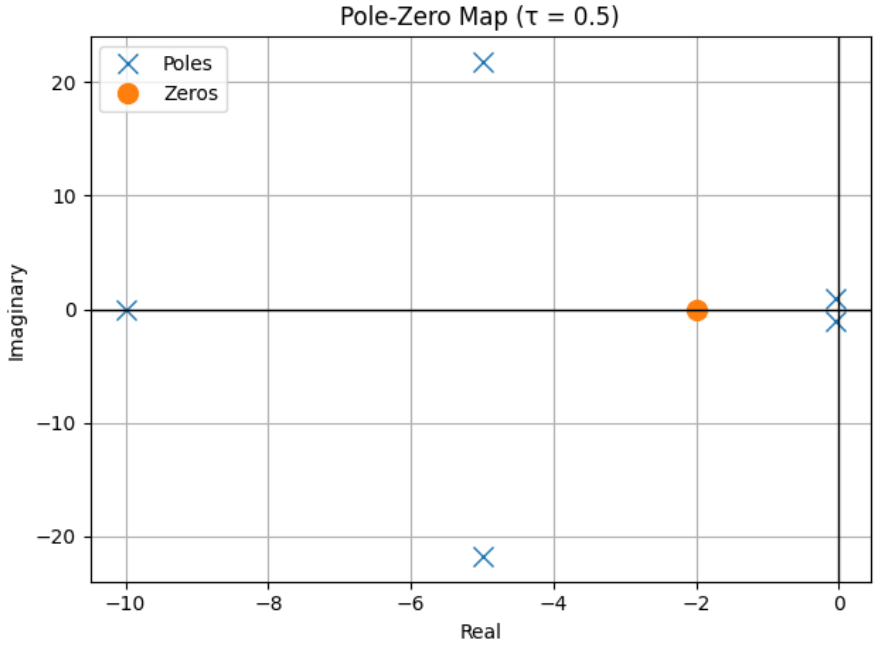

Figure 6: Pole-Zero Map ($\tau = 0.5$).

case $\tau = 20$:

$$G(s) = \frac{20s + 1}{(0.1s + 1)(0.002s^2 + 0.02s + 1)(s^2 + 0.1s + 1)} \tag{10}$$

For equation (10), we have have the following Poles and Zeros:

Poles ($\tau = 20$):

$$\begin{aligned}
(\text{root}_1 \text{ of } 0.002s^2 + 0.02s + 1): &\quad -5.00 + 21.79j \\
(\text{root}_2 \text{ of } 0.002s^2 + 0.02s + 1): &\quad -5.00 - 21.79j \\
(\text{root of } 0.1s + 10): &\quad -10.0 + 00.00j \\
(\text{root}_1 \text{ of } s^2 + 0.1s + 1): &\quad -0.05 + 1.000j \\
(\text{root}_2 \text{ of } s^2 + 0.1s + 1): &\quad -0.05 - 1.000j
\end{aligned}$$

Zeros ($\tau = 20$):

$$[-0.05]$$

Figure 7 illustrates the poles and zeros of the system described by equation (10):

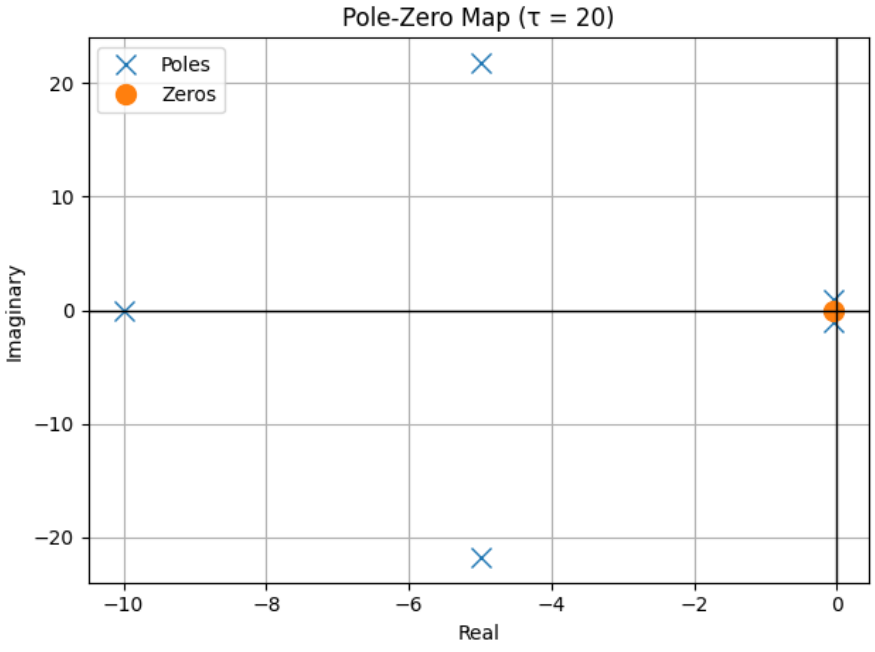

Figure 7: Pole-Zero Map ($\tau = 20$).

ITEM 2.2

According [Ogata, 2010], the zero can be neglected if the ratios of the real parts of the closed-loop poles exceed 5 and there are no zeros nearby, then the closed-loop poles nearest the $j\omega$-axis will dominate in the transient response behavior because these poles correspond to transient-response terms that decay slowly.

case $\tau = 0.5$:

From item 2.1, the *dominant poles* of equation (9) are the complex-conjugate roots of $s^2 + 0.1s + 1$:

$$s1 = -0.05 + 1.00j$$
$$s2 = -0.05 - 1.00j$$

From item 2.1, the ratio between the real part of the zero = 2.0 and the real part of the dominant pole = 0.05 exceeds 5 and there are no zeros nearby, demonstrating that this zero does not make a significant contribution to the system's response and, therefore, can be disregarded when approximating the system to the second-order as (3).

So, we find the equivalent second-order transfer function of equation (9):

$$G(s) = \frac{1}{s^2 + 0.1s + 1} \tag{11}$$

Conclusion: The zero can be neglected in case of $\tau = 0.5$.

case $\tau = 20$:

The *dominant poles* are the same of case $\tau = 0.5$, because the denominator of equation (9) does not depend on τ.

In this case, the ratio between the real part of the zero = 0.05 and the real part of the dominant pole = 0.05 **does not** exceed 5. Additionally, it is not sufficiently close to any of the additional poles, making it impossible to cancel it out with any of them, which indicates that this zero will have a significant influence on the final response.

So, we find the equivalent second-order transfer function of equation (10):

$$G(s) = \frac{20s + 1}{s^2 + 0.1s + 1} \tag{12}$$

Conclusion: The zero **cannot** be neglected in case $\tau = 20$.

ITEM 2.3

Figure 8 illustrates the step responses for system in equation (8) and the second order equivalent one for $\tau = 0.5$:

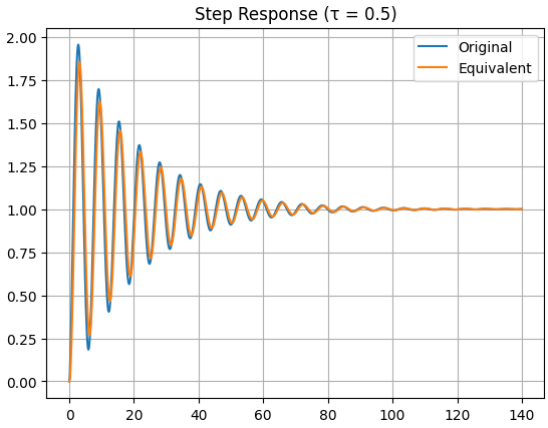

Figure 8: step response ($\tau = 0.5$).

Figure 9 illustrates the step responses for system in equation (8) and the second order equivalent one for $\tau = 20$:

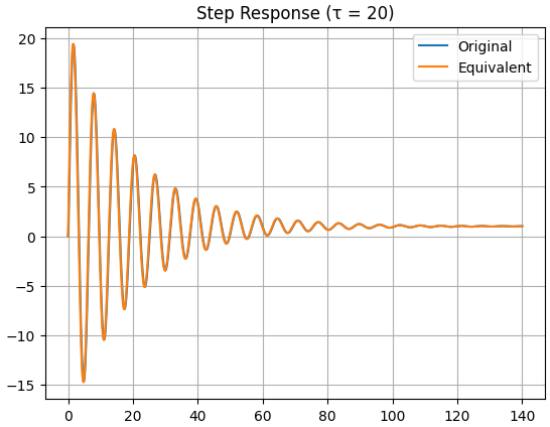

Figure 9: step response ($\tau = 20$).

The difference between these two responses is because, in case $\tau = 0.5$, the zero could be neglected, meaning its removal did not significantly alter the behavior of the function (Figure 8). On the other hand, in case $\tau = 20$, the zero cannot be neglected, and its removal would have significantly affected the behavior of the function, leading to the results seen in Figure 10.

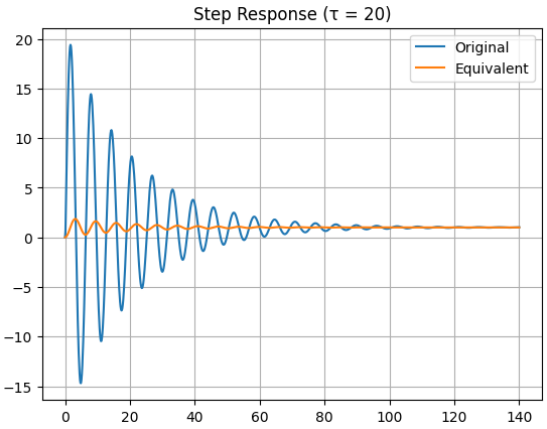

Figure 10: step response ($\tau = 20$).

Exercise 3

It is given the state-space model in (13).

$$\begin{cases} \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ -5 & -2 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 2 \end{bmatrix} u(t) \\ y(t) &= \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} \end{cases} \tag{13}$$

1. Compute and explain the transfer function corresponding to the system in (13).
2. Briefly explain the concept of *bounded-input, bounded-output* (BIBO) stability. Evaluate BIBO stability for the given system.
3. Briefly explain the concept of Lyapunov stability. Evaluate Lyapunov stability for the given system.
4. Discuss how BIBO stability theory and Lyapunov stability differ for LTI systems. Compare and explain the results of item (2) and (3) of this exercise.

Solution

Theoretical Background

Item 3.1

It is possible to write the state equations (related to the system's internal variables) and output equations in vector-matrix form if the system is linear. As [Nise, 2011] states, the state equations can be written as

$$\dot{x} = \mathbf{A}x + \mathbf{B}u$$

and the output equation can be written as show below:

$$y = \mathbf{C}x + \mathbf{D}u$$

where $\mathbf{u}$ represents the input, $\mathbf{x}$ the state variables vector, $\dot{\mathbf{x}}$ the derivative of the state vector with respect to time, and $\mathbf{y}$ the output.

Having a state-space model we can convert it to a transfer function, or the inverse. Given the state and output equations above, we take the Laplace transform and we assume zero initial conditions:

$$s\mathbf{X}(s) = \mathbf{A}\mathbf{X}(s) + \mathbf{B}\mathbf{U}(s)$$
$$\mathbf{Y}(s) = \mathbf{C}\mathbf{X}(s) + \mathbf{D}\mathbf{U}(s)$$

Isolating *X(s)* from the first equation we get:

$$\mathbf{X}(s) = (s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}\mathbf{U}(s) \tag{14}$$

where $\mathbf{I}$ is the identity matrix. And replacing this in output equation, we have the following result:

$$\mathbf{Y}(s) = \left[\mathbf{C}\left(s\mathbf{I} - \mathbf{A}\right)^{-1}\mathbf{B} + \mathbf{D}\right]\mathbf{U}(s)$$

Since $G(s) = Y(s)/U(s)$, the transfer function is:

$$G(s) = \frac{Y(s)}{U(s)} = \left[\mathbf{C}\left(s\mathbf{I} - \mathbf{A}\right)^{-1}\mathbf{B} + \mathbf{D}\right] \tag{15}$$

For the matrixes of the problems, we have:

$$\begin{aligned} G(s) &= \left[C\left(sI - A\right)^{-1}B + D\right] \\ &= \begin{bmatrix}0 & 1\end{bmatrix}\begin{bmatrix}s & -1 \\ 5 & s+2\end{bmatrix}^{-1}\begin{bmatrix}0 \\ 2\end{bmatrix} + \begin{bmatrix}0\end{bmatrix} \\ &= \frac{\left(\begin{bmatrix}0 & 1\end{bmatrix}\begin{bmatrix}s+2 & 1 \\ -5 & s\end{bmatrix}\begin{bmatrix}0 \\ 2\end{bmatrix}\right)}{s^2+2s+5} \\ &= \frac{2s}{s^2+2s+5} \end{aligned}$$

Using *Control*'s function *ss2tf*, it is possible to convert a state-space model to a transfer function. Using the same matrices provided in the exercise, the computational result was the same as the manual (besides floating point errors). The output result was this object representing the transfer function:

```
<TransferFunction>: sys[1]
Inputs (1): ['u[0]']
Outputs (1): ['y[0]']


2 s + 2.665e-15
---------------
 s^2 + 2 s + 5
```

Item 3.2

One of the most important characteristics in a dynamic system is its stability (if a system is stable or not). According [Nise, 2011], a LTI (linear time-invariant) system is *stable* if, when subjected to an initial condition, it comes back to its equilibrium state. It's *marginally stable* if it's response does not decay or grows, but keeps oscillating as time goes by. Finally, a system is *unstable* if the response grows without a bound from equilibrium state.

One way of verifying the stability of a closed-loop system is by checking its BIBO (Bounded-Input Bounded-Output) stability. A system is BIBO stable if its output remains bounded for every bounded input. In other words, if the input's magnitude never exceeds a value M, the output's magnitude must also stay within a certain value N.

For evaluating BIBO stability in system (13), we have to determine all the poles in the $j\omega$-axis. Mathematically, stability for linear, time-invariant systems can be determined from the location of the closed-loop poles:

- If the poles are only in the left half-plane, the system is stable.
- If any poles are in the right half-plane, the system is unstable.
- If the poles are on the $j\omega$-axis and in the left half-plane, the system is marginally stable as long as the poles on the $j\omega$-axis are of unit multiplicity; it is unstable if there are any multiple $j\omega$ poles.

Using the transfer function given by item 3.1:

$$G(s) = \frac{2s}{s^2 + 2s + 5} \tag{16}$$

we get the following poles: $-1 + 2j$, $-1 - 2j$

Since all the poles are in the left half-plane of the $j\omega$-axis, the system (13) is **stable**.

Item 3.3

BIBO stability, while being one of the most classic methods for analyzing stability, is limited to linear and time-invariant systems. However, for more complex systems, especially nonlinear ones, another method called internal stability (or Lyapunov stability) can be used.

While BIBO stability analyzes a system's stability based on its input-output (I/O) behavior, Lyapunov stability evaluates it through the system's state-space representation. This makes Lyapunov stability a more versatile approach, as it can be applied to both linear and nonlinear systems.

Consider the following autonomous state equation (i.e., it is not disturbed by any input):

$$\dot{x} = Ax \tag{17}$$

We want to know how the states of the system evolve over time from an initial condition. We say that the system is marginally stable or stable in the sense of Lyapunov if and only if every finite initial state produces a bounded response. In other words, when this system is released from an initial condition, it will produce a bounded response. A system is asymptotically stable if, in addition to being stable in the sense of Lyapunov, the system eventually returns to the equilibrium state as time tends to infinity.

In addition, the stability of a linear system can be assessed using the eigenvalues of its system matrix (denoted as A). The eigenvalues give information about the system's dynamic behavior.

- If all eigenvalues of the system matrix A have negative real parts, the system is asymptotically stable.
- If any eigenvalue has a positive real part, the system is unstable.
- If any eigenvalue has a real part equal to zero, the system is marginally stable, and further analysis is needed.

To calculate the eigenvalues of the matrix A, we need to find the roots of the characteristic equation:

$$\det(A - \lambda I) = 0 \tag{18}$$

Where the eigenvalues λ of the matrix A are the solutions to the characteristic equation and I is the identity matrix of the same size as A.

So, $A - \lambda I$:

$$A - \lambda I = \begin{bmatrix} 0 & 1 \\ -5 & -2 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} = \begin{bmatrix} 0 - \lambda & 1 \\ -5 & -2 - \lambda \end{bmatrix} = \begin{bmatrix} -\lambda & 1 \\ -5 & -2 - \lambda \end{bmatrix}$$

Now, we have to compute the determinant of the matrix $A - \lambda I$:

$$\det(A - \lambda I) = \det\left(\begin{bmatrix} -\lambda & 1 \\ -5 & -2 - \lambda \end{bmatrix}\right)$$

For a 2×2 matrix, the determinant is calculated as:

$$\det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc$$

Applying this to our matrix:

$$\det(A - \lambda I) = (-\lambda)(-2 - \lambda) - (1)(-5) = \lambda^2 + 2\lambda + 5$$

Solving the characteristic equation $\lambda^2 + 2\lambda + 5 = 0$ we have the eigenvalues:

$$\lambda_1 = -1 + 2i \quad \text{and} \quad \lambda_2 = -1 - 2i$$

Since both eigenvalues real parts are negative, the system (13) is asymptotically stable.

These eigenvalues indicate that the system exhibits oscillatory behavior due to the imaginary components, but the oscillations will decay over time because of the negative real parts.

Item 3.4

BIBO stability focuses on the external characteristics of a system, examining whether bounded inputs produce bounded outputs. For LTI systems, BIBO stability is closely tied to the absolute convergence of the system's impulse response and is often analyzed using transfer functions. On the other hand, Lyapunov stability takes a more general approach, concentrating on the internal state of the system and the behavior of state trajectories over time. It employs energy-like functions known as Lyapunov functions and is applicable to both linear and nonlinear systems.

While BIBO stability is specific to input-output systems, Lyapunov stability is more versatile, applicable even to systems without explicit inputs or outputs. BIBO stability requires knowledge of the system's transfer function or impulse response, whereas Lyapunov stability needs a suitable Lyapunov function, which can sometimes be challenging to determine.

A key concept in stability analysis for LTI systems is asymptotic stability. A system is considered asymptotically stable if and only if all its poles (or eigenvalues of the state matrix A) have negative real parts. This condition ensures that any perturbation in the system will decay over time, returning the system to its equilibrium state.

Lyapunov's theorem provides an alternative method to assess stability without directly calculating eigenvalues. For continuous-time systems, the theorem states that if the eigenvalues of matrix A have negative real parts, then for any symmetric positive definite matrix Q, there exists a unique solution P to the Lyapunov equation:

$$A^T P + PA = -Q \tag{19}$$

where P is also symmetric and positive definite. This theorem complements both eigenvalue analysis and Lyapunov's direct method, being useful for stability analysis (especially for complex systems where eigenvalues cannot be easily found.)

BIBO and Lyapunov stability analyses are valuable tools for understanding LTI systems. Complementing each other, they provide a wide view of system stability from both external and internal perspectives.

for LTI systems, there's a relationship between these stability concepts: if an LTI system is BIBO stable, it is also Lyapunov stable, it can be seen in items 3.2 and 3.3, since the LTI system is BIBO stable (all the poles in the left half-plane of the $j\omega$-axis) and it also is Lyapunov stable (all the eighenvalues have negative real parts). Though, the converse is not always true. This relationship underscores the interconnected nature of different stability concepts in control theory.

Exercise 4

The unity feedback system has the forward transfer function given by (20):

$$G(s) = \frac{K(s+7)}{s(s^3 + 25s^2 + 196s + 480)} \tag{20}$$

1. Evaluate the system type.
2. Find the value of K to yield a 1% error in steady-state for an input of $0.1t$.
3. Find the static error constants for the value of K found in item () of this exercise.
4. Verify the stability of your system and plot its step response.

Solution

Theoretical Background

As [Nise, 2011] defines, *steady-state error* is the difference between the input and the output for a prescribed test input as $t \to \infty$. Test inputs types include step, ramp, and parabola. These errors can be caused by the system's configuration itself, and the type of input can also affect the error. For example, a unit step input may result in a much smaller error than a parabola, depending on the system.

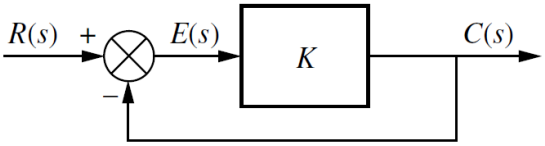

Figure 11: Steady-state error for unity feedback system.

Consider Figure 11, where $C(s)$ is the system's output, $R(s)$ is the input, and $E(s)$ is the error as the difference between the input and the output, in addition to a block with a pure gain K. Considering the input as the unit step, we see from the system that

$$C(s) = KE(s) \tag{21}$$

Isolating the error and marking with a subscript $_{steady-state}$ on the error and the output to indicate the system in steady state, we have:

$$e_{steady-state} = \frac{1}{K} c_{steady-error} \tag{22}$$

From Equation 22, it is clear that a smaller gain K will result in a larger error.

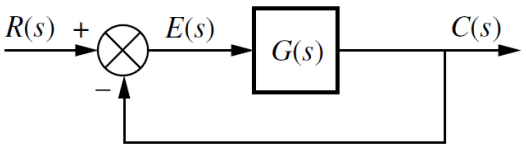

Figure 12: Closed-loop control system error with feedback.

Using the figure 12 which has a unit feedback and a transfer function $G(s)$, solving for $E(s)$, we get:

$$E(s) = \frac{R(s)}{1 + G(s)} \tag{23}$$

Since we are interested in the steady-state error, we will apply the final value theorem by taking the limit as $t \to \infty$. Since $t \to \infty$ corresponds to $s \to 0$ in the frequency domain,

we have:

$$e(\infty) = \lim_{s\to 0} sE(s) = \lim_{s\to 0} \frac{sR(s)}{1+G(s)} \tag{24}$$

Now, we will analyze how the error behaves for test inputs (step, ramp, and parabola).

- **Unit Step**: Substituting the unit step $(1/s)$ as the input $R(s)$ into equation 24, we get:

$$e(\infty) = \lim_{s\to 0} \frac{s\frac{1}{s}}{1+G(s)} = \lim_{s\to 0} \frac{1}{1+G(s)} \tag{25}$$

  We see that the limit will only apply to the transfer function $G(s)$, and to make the error approach zero, we need $\lim_{s\to 0} G(s) = \infty$.

- **Ramp**: Substituting the ramp $(1/s^2)$ as the input $R(s)$ into equation 24, we get:

$$e(\infty) = \lim_{s\to 0} \frac{s\frac{1}{s^2}}{1+G(s)} = \lim_{s\to 0} \frac{1}{s+sG(s)} = \lim_{s\to 0} \frac{1}{sG(s)} \tag{26}$$

  In this case, to make the error approach zero, we need $\lim_{s\to 0}\ sG(s) = \infty$.

- **Parabola**: Substituting the parabola $(1/s^3)$ as the input $R(s)$ into equation 24, we get:

$$e(\infty) = \lim_{s\to 0} \frac{s\frac{1}{s^3}}{1+G(s)} = \lim_{s\to 0} \frac{1}{s^2+s^2G(s)} = \lim_{s\to 0} \frac{1}{s^2G(s)} \tag{27}$$

  In this case, to make the error approach zero, we need $\lim_{s\to 0}\ s^2G(s) = \infty$.

Analyzing the three cases above, we see that the determination of the error always depends on the term in the denominator. This term, which we rely on to determine the error in the steady state, is called the static error constant.

- **Position constant (K_p)**:

$$K_p = \lim_{s\to 0} G(s) \tag{28}$$

- **Velocity constant (K_v)**:

$$K_v = \lim_{s\to 0} sG(s) \tag{29}$$

- **Acceleration constant (K_a)**:

$$K_a = \lim_{s\to 0} s^2G(s) \tag{30}$$

In addition to the static error constants, there are also the system types. Consider the following transfer function with unit feedback:

$$G(s) = \frac{K(T_a s + 1)(T_b s + 1) \cdots (T_m s + 1)}{s^n (T_1 s + 1)(T_2 s + 1) \cdots (T_p s + 1)} \tag{31}$$

[Ogata, 2010] explains that the term s^n represents a pole of multiplicity "n" at the origin, and that this classification of systems is based on the number of integrations indicated by the open-loop transfer function. Thus, for $n = 0$, $n = 1$, $n = 2$, and so on, we say that the systems are of type 0, 1, and 2, respectively.

		Type 0		Type 1		Type 2	
Input	**Steady-state error formula**	**Static error constant**	**Error**	**Static error constant**	**Error**	**Static error constant**	**Error**
Step, $u(t)$	$\frac{1}{1+K_p}$	$K_p = \text{Constant}$	$\frac{1}{1+K_p}$	$K_p = \infty$	0	$K_p = \infty$	0
Ramp, $tu(t)$	$\frac{1}{K_v}$	$K_v = 0$	∞	$K_v = \text{Constant}$	$\frac{1}{K_v}$	$K_v = \infty$	0
Parabola, $\frac{1}{2}t^2u(t)$	$\frac{1}{K_a}$	$K_a = 0$	∞	$K_a = 0$	∞	$K_a = \text{Constant}$	$\frac{1}{K_a}$

Figure 13: System types and static error constants.

These constants can be used to specify the steady-state error of a system. Considering an arbitrary $K_i = n$ (finite), the first thing to note is if the system is stable, as we can discuss steady-state error only in stable systems. Another important point is that, by analyzing Figure 13, we can determine the type of system and the applied input, as well as calculate the error based on the given K_i.

ITEM 4.1

To evaluate the system type we have to analyze the number of pure integrations in the forward path, equivalently, the value of "n" in the denominator of equation 31

$$G(s) = \frac{K(s+7)}{s(s^3+25s^2+196s+480)}$$

Since "n" $= 1$ in equation 20, the system is a type 0 system.

ITEM 4.2

To find the value of K that yields a 1% error in steady-state for an input of $0.1t$, we use equation 26 with an simple modification, since we have to modify the amplitude of the ramp input from 1.0 to 0.1:

$$e(\infty) = \lim_{s\to 0} \frac{0.1}{sG(s)} \tag{32}$$

From equation (20) and equation (32), we have:

$$0.01 = \lim_{s\to 0} \frac{0.1(s^3+25s^2+196s+480)}{K(s+7)}$$

Applying the limit and solving, we have $K = 685.7142857142853$.

ITEM 4.3

To find the static error constants for the value of K found in item 4.2 of this exercise we will use the equations (28), (29) and (30).

- **Position constant (K_p)**:

$$K_p = \lim_{s\to 0} G(s) = \lim_{s\to 0} \frac{685.714285714(s+7)}{s(s^3+25s^2+196s+480)} = \infty$$

- **Velocity constant (K_v)**:

$$K_v = \lim_{s\to 0} sG(s) = \lim_{s\to 0} \frac{685.714285714(s+7)}{s^3+25s^2+196s+480} = 10$$

- **Acceleration constant (K_a)**:

$$K_a = \lim_{s\to 0} s^2G(s) = \lim_{s\to 0} \frac{s(685.714285714)(s+7)}{s^3+25s^2+196s+480} = 0$$

ITEM 4.4

To verify the stability of our system and plot its step response, first we have to simplify the feedback in the system. For this, we will use equation (21), equation (23) and $G(s)$ in equation (20), leading to the following result:

$$\frac{C(s)}{R(s)} = \frac{G(s)}{1+G(s)} = \frac{K(s+7)}{s(s^3+25s^2+196s+480)+K(s+7)}$$

And using K (amplitude) from item 3.2, finally:

$$\frac{C(s)}{R(s)} = \frac{685.7s+4800}{s^4+25s^3+196s^2+1166s+4800} \tag{33}$$

For equation (33), we have have the following Poles:

$$\begin{aligned} -16.22371199 + 0.j \\ -7.09614714 + 0.j \\ -0.84007044 + 6.4021704j \\ -0.84007044 - 6.4021704j \end{aligned}$$

Figure 14 illustrates the poles and zeros of the system described by equation (9):

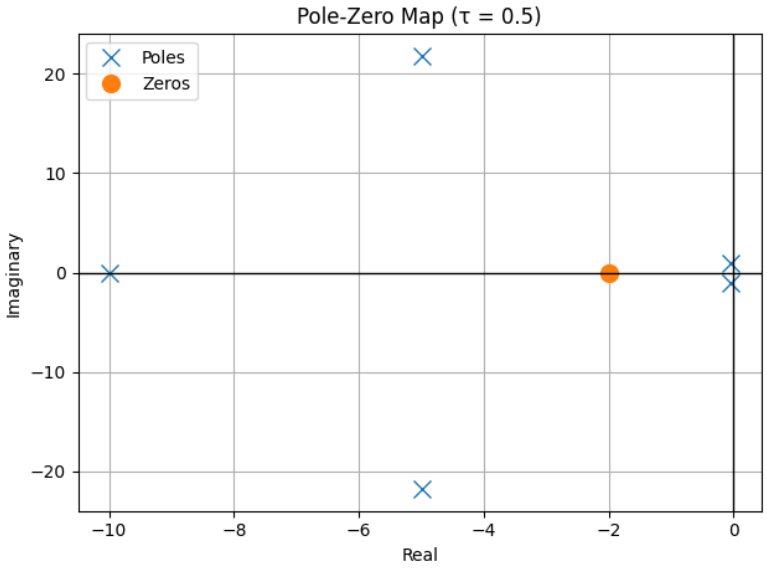

Figure 14: Pole-Zero Map ($\tau = 0.5$).

Since all the poles are in the left half-plane of the $j\omega$-axis, the system (33) is **stable**.

Figure 15 illustrates the step response in the feedback system (33), which confirms the stabilization of the system:

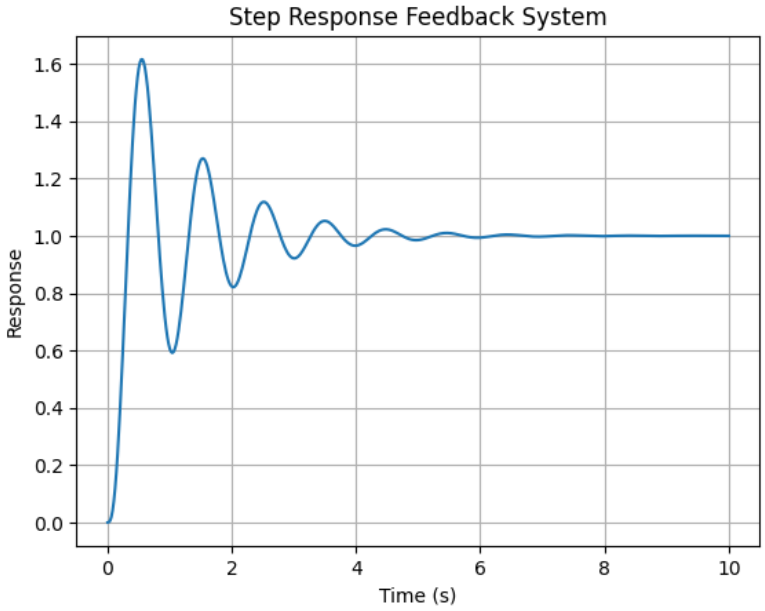

Figure 15: step response feedback system.

Appendix

Exercise 1

Item 1.1

Listing 1: Item 1.1

```
!pip install control
import control as ctrl
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import signal
from scipy.linalg import solve_continuous_lyapunov
from sympy import symbols, limit, solve, latex
from IPython.display import display, Math

data_a = pd.read_csv('HW2_ex1_dataA.csv', header=None, names=["time", "
    response"])
data_b = pd.read_csv('HW2_ex1_dataB.csv', header=None, names=["time", "
    response"])

plt.figure(figsize=(10, 5))
plt.plot(data_a["time"], data_a["response"])
plt.xlabel('t')
plt.ylabel('y(t)')
plt.title('Step response (data A)')
plt.grid()
plt.show()

plt.figure(figsize=(10,5))
plt.plot(data_b["time"], data_b["response"])
plt.xlabel('t')
plt.ylabel('y(t)')
plt.title('Step response (data B)')
plt.grid()
plt.show()
```

Item 1.2

Listing 2: Item 1.2

```
def find_settling_time_index(df, range_value, steady_state):
  for i in range(len(df) - 1, 0, -1):
    if not (steady_state - range_value <= df["response"].iloc[i-1] <=
        steady_state + range_value):
      return i
  return 0

# For system A
steady_state_a = data_a["response"].iloc[-1]
```

```
# Time constant calculation (63.2% criterion)
sixty_three_perc = 0.632 * steady_state_a
time_constant_index = data_a.index[(data_a["response"] >=
    sixty_three_perc)].to_list()[0]
time_constant_a = data_a["time"].loc[time_constant_index]

# Rise time calculation (10% and 90% criterion)
ten_perc = 0.1 * steady_state_a
ninety_perc = 0.9 * steady_state_a
time_at_10_perc_a = data_a.loc[data_a["response"] >= ten_perc, "time"].
    iloc[0]
time_at_90_perc_a = data_a.loc[data_a["response"] >= ninety_perc, "time
    "].iloc[0]
rise_time_a = time_at_90_perc_a - time_at_10_perc_a

# Settling time calculation (2% criterion)
two_perc_range = 0.02 * steady_state_a
settling_time_index = find_settling_time_index(data_a, two_perc_range,
    steady_state_a)
settling_time_a = data_a["time"].loc[settling_time_index]

# Transfer Function
num = [steady_state_a]
den = [time_constant_a, 1]
tf_a = signal.TransferFunction(num, den)

print(f"Time constant: {time_constant_a}")
print(f"Rise time: {rise_time_a}")
print(f"Settling time: {settling_time_a}")
print(f"Transfer Function: {tf_a}")
ctrl.step_info(data_a["response"],T=data_a["time"])

# For System B
# Max and steady state values
max = data_b["response"].max()
steady_state_b = data_b["response"].iloc[-1]

# Peak time
peak_index_b = data_b["response"].idxmax()
peak_time_b = data_b["time"].loc[peak_index_b]

# Overshoot and Damping ratio (zeta) calculation
overshoot = ((max - steady_state_b) / steady_state_b)
ln_overshoot = np.log(overshoot)
zeta = -ln_overshoot / np.sqrt(np.pi**2 + ln_overshoot**2)

# Natural frequency (Omega n)
wn = np.pi / (peak_time_b * np.sqrt(1 - zeta**2))

# Settling time calculation (2% criterion)
two_perc_range = 0.02 * steady_state_b
settling_time_index = find_settling_time_index(data_b, two_perc_range,
    steady_state_b)
settling_time_b = data_b["time"].loc[settling_time_index]
```

```
# Rise time calculation (10% and 90% criterion)
ten_perc = 0.1 * steady_state_b
ninety_perc = 0.9 * steady_state_b
time_at_10_perc_b = data_b.loc[data_b["response"] >= ten_perc, "time"].
    iloc[0]
time_at_90_perc_b = data_b.loc[data_b["response"] >= ninety_perc, "time
    "].iloc[0]
rise_time_b = time_at_90_perc_b - time_at_10_perc_b

# Transfer function
num = [steady_state_b * wn**2]
den = [1, 2*zeta*wn, wn**2]
tf_b = signal.TransferFunction(num, den)

print(f"max: {max}")
print(f"peak_time_b: {peak_time_b}")
print(f"steady_state_b: {steady_state_b}")
print(f"Damping ratio (zeta): {zeta}")
print(f"Natural frequency (wn): {wn}")
print(f"Rise time: {rise_time_b}")
print(f"Peak time: {peak_time_b}")
print(f"Overshoot: {overshoot}")
print(f"Settling time: {settling_time_b}")
print(f"Transfer Function: {tf_b}")
ctrl.step_info(data_b["response"],T=data_b["time"])
```

Item 1.3

Listing 3: Item 1.3

```
t = np.linspace(0, data_a["time"].iloc[-1], 1000)
t, y = signal.step(tf_a, T=t)

plt.figure(figsize=(10, 5))
plt.plot(data_a["time"], data_a["response"], label='original')
plt.plot(t, y, label='simulation')
plt.axvline(x=time_constant_a, color='b', linestyle='--', linewidth=1,
    label='time constant')
plt.axvline(x=time_at_10_perc_a, color='g', linestyle='--', linewidth=1,
    label='10% rise start')
plt.axvline(x=time_at_90_perc_a, color='m', linestyle='--', linewidth=1,
    label='90% rise end')
plt.axvline(x=settling_time_a, color='r', linestyle='--', linewidth=1,
    label='settling time')
plt.xlabel('t')
plt.ylabel('y(t)')
plt.title('Step response (data A)')
plt.legend()
plt.grid()
plt.show()

t = np.linspace(0, data_b["time"].iloc[-1], 128)
```

```
t, y = signal.step(tf_b, T=t)

# Plot the response
plt.figure(figsize=(10, 5))
plt.plot(data_b["time"], data_b["response"], label='original')
plt.plot(t, y, label='simulated')
plt.axvline(x=settling_time_b, color='r', linestyle='--', linewidth=1,
    label='settling time')
plt.axvline(x=peak_time_b, color='b', linestyle='--', linewidth=1, label=
    'peak time')
plt.axvline(x=time_at_10_perc_b, color='g', linestyle='--', linewidth=1,
    label='10% rise start')
plt.axvline(x=time_at_90_perc_b, color='m', linestyle='--', linewidth=1,
    label='90% rise end')
plt.title('Step response (data B)')
plt.xlabel('t')
plt.ylabel('y(t)')
plt.grid(True)
plt.legend()
plt.show()
```

Exercise 2

Item 2.1

Listing 4: Item 2.1

```
def print_poles(poles):
    for pole in poles:
        real = pole.real
        imag = pole.imag

        if imag == 0:
            print(f"  {real:.2f}")
        elif imag > 0:
            print(f"  {real:.2f} + {imag:.2f}j")
        else:
            print(f"  {real:.2f} - {abs(imag):.2f}j")

def analyze_system(tau):
  num = [tau, 1]
  den = np.convolve([0.1, 1],
        np.convolve([0.002, 0.02, 1], [1, 0.1, 1]))

  G = signal.TransferFunction(num, den)

  # Step response
  t, y = signal.step(G, T=np.linspace(0, 140, 1000))

  # Pole-zero map
  poles = np.roots(den)
  zeros = np.roots(num)

  # Pole-zero map
  plt.plot(np.real(poles), np.imag(poles), 'x', markersize=10, label='
      Poles')
  plt.plot(np.real(zeros), np.imag(zeros), 'o', markersize=10, label='
      Zeros')
  plt.title(f'Pole-Zero Map (   = {tau})')
  plt.xlabel('Real')
  plt.ylabel('Imaginary')
  plt.grid(True)
  plt.legend()
  plt.axhline(y=0, color='k', linewidth=1)
  plt.axvline(x=0, color='k', linewidth=1)

  plt.tight_layout()
  plt.show()

  # Print poles and zeros
  print(f"Poles (   = {tau}):")
  print_poles(poles)
  print(f"\nZeros (   = {tau}):")
  print(zeros)
```

```
  return {
      "tau": tau,
      "time": t,
      "response": y,
      "poles": poles,
      "zeros": zeros
  }

# Analyze for    = 0.5
analysis_05 = analyze_system(0.5)

# Analyze for    = 20
analysis_20 = analyze_system(20)
```

Item 2.2

Listing 5: Item 2.2

```
def find_dominant_poles(poles):
  # Dominant poles are those with the smallest real parts (closest to the
      imaginary axis)
  sorted_poles = sorted(poles, key=lambda p: np.abs(np.real(p)))
  return sorted_poles[:2]

def second_order_approximation(dominant_poles):
  s1, s2 = dominant_poles # typically complex conjugates
  wn = np.sqrt(np.real(s1)**2 + np.imag(s1)**2) # natural frequency

  num = [wn ** 2]
  den = [1, 2 * (-np.real(s1) / wn) * wn, wn ** 2]
  tf = signal.TransferFunction(num, den)

  return wn, -np.real(s1) / wn, tf

def analyze_equivalent_second_order_system(tau, analysis):
  poles = analysis["poles"]
  zeros = analysis["zeros"]

  dominant_poles = find_dominant_poles(poles)
  wn, zeta, tf = second_order_approximation(dominant_poles)

  print(f"Tau: {tau}")
  print(f"Dominant Poles: {dominant_poles}")
  print(f"Natural Frequency ( _n ): {wn}")
  print(f"Damping Ratio (  ): {zeta}\n")

  print(f"Transfer Function: {tf}\n")

  # Check if the zero can be neglected
  zero = zeros[0]

  print(f"Real part of the zero: {np.real(zero)}")
```

```
  print(f"Real part of the dominant pole * 5: {np.real(dominant_poles[0])
      * 5}\n")

  if np.abs(np.real(zero)) > 5 * np.abs(np.real(dominant_poles[0])):
    print("The zero can be neglected in this case.")
  else:
    print("The zero cannot be neglected in this case.")

  return tf

tf_05 = analyze_equivalent_second_order_system(0.5, analysis_05)
tf_20 = analyze_equivalent_second_order_system(20, analysis_20)
```

Item 2.3

Listing 6: Item 2.3

```
t, y = signal.step(tf_05, T=analysis_05["time"])

plt.plot(analysis_05["time"], analysis_05["response"], label='Original')
plt.plot(t, y, label='Equivalent')
plt.title(f'Step Response (tau = {0.5})')
plt.legend()
plt.grid()
plt.show()

# num = [1]
# den = [1, 0.1, 1]
# tf05 = signal.TransferFunction(num, den)

# t, y = signal.step(tf05, T=analysis_05["time"])

# plt.plot(analysis_05["time"], analysis_05["response"], label='Original'
    )
# plt.plot(t, y, label='Equivalent')
# plt.title(f'Step Response (tau = {0.5})')
# plt.legend()
# plt.grid()
# plt.show()

num = [20, 1]
den = [1, 0.1, 1]
tf20 = signal.TransferFunction(num, den)

t, y = signal.step(tf20, T=analysis_20["time"])

plt.plot(analysis_20["time"], analysis_20["response"], label='Original')
plt.plot(t, y, label='Equivalent')
plt.title(f'Step Response (tau = {20})')
plt.legend()
plt.grid()
plt.show()
```

```
# t, y = signal.step(tf_20, T=analysis_20["time"])

# plt.plot(analysis_20["time"], analysis_20["response"], label='Original'
    )
# plt.plot(t, y, label='Equivalent')
# plt.title(f'Step Response (ta = {20})')
# plt.legend()
# plt.grid()
# plt.show()
```

Exercise 3

Item 3.1

Listing 7: Item 3.1

```
A = np.array([[0, 1], [-5, -2]])
B = np.array([[0], [2]])
C = np.array([[0, 1]])
D = np.array([[0]])
tf = ctrl.ss2tf(A, B, C, D)
print(tf)
```

Item 3.2

Listing 8: Item 3.2

```
poles = tf.poles()
print(f"Poles: {poles}")
bibo_stable = all(np.real(poles) < 0)
print(f"BIBO Stable: {bibo_stable}")
```

Item 3.3

Listing 9: Item 3.3

```
eigenvalues = np.linalg.eigvals(A)
print(f"Eigenvalues: {eigenvalues}") # identical to the system poles
lyapunov_stable = all(np.real(eigenvalues) < 0)
print(f"Lyapunov Stable: {lyapunov_stable}")
```

Item 3.4

Listing 10: Item 3.4

```
# continuous problem: A.T * P + P*A = -Q
Q = np.eye(2)
P = solve_continuous_lyapunov(A.T, -Q)

# Check if P is positive definite
is_positive_definite = np.all(np.linalg.eigvals(P) > 0)

print("P matrix:")
print(P)
print(f"Is P symmetric? {np.allclose(P, P.T)}")
print(f"Is P positive definite? {is_positive_definite}")
```

Exercise 4

Item 4.1

Listing 11: Item 4.1

```
# Type 1: one pole at origin
num = [1, 7]
den = np.convolve([1, 0], [1, 25, 196, 480])
G = signal.TransferFunction(num, den)
print(f"Poles: {G.poles}")
```

Item 4.2

Listing 12: Item 4.2

```
s, K = symbols('s K')

A = 0.1  # Slope of the ramp input
ess = 0.01  # Desired steady-state error

G_s = K * (s + 7) / (s * (s**3 + 25*s**2 + 196*s + 480))

Kv = A / ess
Kv_eq = limit(s * G_s, s, 0)
eq = Kv - Kv_eq

K_value = solve(eq, K)[0]

display(Math('K = ' + latex(K_value)))

# Nise ch7p2 (Appendix B)

# Define numerator of G(s)/K.
numgdK = np.array([1, 7])

# Define denominator of G(s)/K.
dengdK = np.convolve([1, 0], [1, 25, 196, 480])

# Create G(s)/K.
GdK = ctrl.TransferFunction(numgdK, dengdK)

# Define numerator of sG(s)/K.
numgkv = np.convolve([1, 0], numgdK)

# Define denominator of sG(s)/K.
dengkv = dengdK

# Create sG(s)/K.
GKv = ctrl.TransferFunction(numgkv, dengkv)

# Cancel common 's' in numerator and denominator of sG(s)/k.
GKv = ctrl.minreal(GKv, verbose=False)
```

```
# Evaluate (Kv/K) = (numgkv/dengkv) for s=0.
KvdK = ctrl.dcgain(GKv)

# Enumerate steady-state error.
ess = 0.01

# Solve for K.
K_value = 0.1 / (ess * KvdK)
print(f"K = {K_value}")
```

Item 4.3

Listing 13: Item 4.3

```
# G_s = K * (s + 7) / (s * (s**3 + 25*s**2 + 196*s + 480))
G_s = G_s.subs({K: K_value})

K_p = limit(G_s, s, 0)
K_v = limit(s * G_s, s, 0)
K_a = limit(s**2 * G_s, s, 0)

display(Math('K_p = ' + latex(K_p)))
display(Math('K_v = ' + latex(K_v)))
display(Math('K_a = ' + latex(K_a)))
```

Item 4.4

Listing 14: Item 4.4

```
# Nise ch7p2 (Appendix B)
# Check stability

# Form T(s).
T = ctrl.feedback(K_value * GdK, 1)

# Display closed-loop poles.
poles = ctrl.poles(T)
print(f"Closed-loop poles = {poles}")

t, y = ctrl.step_response(T, T=np.linspace(0, 10, 500))
plt.plot(t, y)
plt.title('Step Response Feedback System')
plt.xlabel('Time (s)')
plt.ylabel('Response')
plt.grid(True)
plt.show()
```

References

[Nise, 2011] Nise, N. (2011). *Control Systems Engineering, Sixth.* John Wiley & Sons, Incorporated.

[Ogata, 2010] Ogata, K. (2010). *Modern Control Engineering.* Prentice Hall.